



Triakis Corporation

System Test Design Document

For the

Shuttle Remote Manipulator System

**A NASA CI03
SARP Initiative 583
IVV-70 Project**



Table of Contents

1	Introduction.....	3
1.1	Purpose.....	3
1.2	References	3
2	System Test Design	3
2.1	Automatic Test Program.....	3
2.1.1	ES Automatic Tests.....	4
2.1.2	DE Automatic Tests.....	5
2.2	System Requirements Test Program	10
2.2.1	Fault reporting tests	10
2.2.2	RMA joint control tests.....	11
2.2.3	Camera motor control tests	11

Table of Tables

Table 1:	ES Automatic Test Program.....	4
Table 2:	DE Automatic Test Program.....	6
Table 3:	Example of Fault Reporting Verification Test	10
Table 4:	SRMS System Requirements Test Program.....	12



1 Introduction

This specification is being developed to support a research project funded by the NASA Software Assurance Research Program (SARP) during the fiscal year 2003 Center Initiatives (CI03) effort. A system-level, executable specification (ES) based simulation of the Shuttle Remote Manipulator System (SRMS) has been created from the requirements specified in the System Requirements (SARP-I583-001) and Simulator Requirements (SARP-I583-002) Specifications, and will be used as a vehicle for exploring the concepts described in section 2 of Triakis proposal number TC_G020614.

This document provides information on the design of the tests used verify the specified SRMS behavior. Rather than a detailed description, this document essentially comprises the code listing of the automatic and system requirements test files. For a real-world project, this document might include more supporting discussion and traceability to the requirements. Since the RMS Computer functional requirements are specified in executable form, traceability may be satisfied through an analysis of the ES code coverage.

The same tests developed to verify the system requirements are also used to verify the executable software running in the DE. The tests are presented in this document and the results are presented in the System Test Document (SARP-I583-302). This research effort does not require complete test coverage in order to achieve the stated goals. The results given in the System Test Document (SARP-I583-302) will show incomplete code test coverage.

1.1 Purpose

The purpose of the virtual system being tested is to facilitate the research goals stated in Triakis proposal number TC_G020614. System components and functions of the real-world SRMS that are not required to support our research goals have been omitted. Please refer to the System Requirements Specification (SARP-I583-001) for a detailed explanation of the project purpose.

1.2 References

- SARP-I583-001 System Requirements Specification for the Shuttle Remote Manipulator System
- SARP-I583-002 Simulator Requirements Specification for the Shuttle Remote Manipulator System
- SARP-I583-302 System Test Document for the Shuttle Remote Manipulator System
- TC_G020614 Triakis proposal to NASA for the SARP (Solicitation No: NRA SARP 0201), 14 June 2002

2 System Test Design

Testing of the SRMS in the simulator is accomplished through the use of two individual executable test programs: Auto.cpp and SystemReqTest.cpp. The Auto.cpp file is run automatically upon completion of loading and running the simulator. The following sections describe the purpose and use of these two programs for testing in the simulator environment.

2.1 Automatic Test Program

The Auto.cpp automatic test program is used as the entry point for testing the simulator. Typically, this program will set up the initial conditions for the simulator and execute one or more programs to perform the desired tests (e.g. system requirements tests, software requirements tests, etc.). In a case such as this project where there is first an ES and then a DE substitution, two different Auto.cpp programs will be created. While the ES version is concerned primarily with running the requirements tests, the DE version is also concerned with lower level tests and metrics gathering. Test results are documented in the System Test Document (SARP-I583-302).

2.1.1 ES Automatic Tests

As discussed in the previous section, the ES version of the Auto.cpp program simply sets up the SRMS initial conditions and calls the SystemRequirementsTest() function (SystemRequirementsTest.cpp file).

The complete code listing for the ES Auto.cpp file may be found in Table 1.

Table 1: ES Automatic Test Program

ES Automatic Test Program

```
*****  
*  
* File: Auto.cpp  
*  
* Description: This file contains the AutoTest function where all simulation testing begins.  
* This file is to be modified by the user to add automatic software tests  
* beyond the scope of the system requirements.  
*  
*****/  
  
#include "Auto.h"  
  
#include "Pointers.h"  
  
#include "OScope.h"  
#include "SigGen.h"  
  
#include "SSGlobalData.h"  
  
void SystemRequirementsTest();  
  
void AutoTest()  
{  
    Message("Hello World");  
  
    InitPointers(); // Init pointers to all parts for use in automatic testing  
  
    World *world = (World *)Sim::ExtNameToPartPtr("The World");  
  
    char sysName[80];  
    strcpy(sysName, world->GetSystemName());  
    Message("System Name:");  
    Message(sysName);  
  
    // Uncomment to Generate "sim" files.  
    //BOOL success = world->GenerateSimFiles(".\\GenSimFiles", TRUE);  
  
    // Uncomment to Generate pointers files  
    //Sim::CreatePartPointerFiles();  
  
    world->RestoreWindows("init.stu");  
    world->SetAutoSpeed(FALSE);
```



ES Automatic Test Program

```
world->SetSpeed(12);

pivot = 90.0;
ang_shoulder = 90.0;
ang_elbow = 90.0;
ang_wristx = 90.0;
ang_wristy = 90.0;
ang_wristz = 90.0;
rot_wrist = 90.0;

pitch_bayaft_cam = 90.0;
yaw_bayaft_cam = 90.0;
zoom_bayaft_cam = 90.0;

pitch_bayfore_cam = 90.0;
yaw_bayfore_cam = 90.0;
zoom_bayfore_cam = 90.0;

pitch_lowerarm_cam = 90.0;
yaw_lowerarm_cam = 90.0;
zoom_lowerarm_cam = 90.0;

pitch_upperarm_cam = 90.0;
yaw_upperarm_cam = 90.0;
zoom_upperarm_cam = 90.0;

pitch_wrist_cam = 90.0;
yaw_wrist_cam = 90.0;
zoom_wrist_cam = 90.0;

eq.Delay(0.2);

// Uncomment to run System Requirements Test
SystemRequirementsTest();

for(;;)
    eq.Delay(1.0);
}
```

2.1.2 DE Automatic Tests

As discussed in the section 2.1, the DE version of the Auto.cpp program performs the following sequence of events:

- a. Set up the SRMS initial conditions.
- b. Set up conditions for gathering code execution metrics.
- c. Call the SystemRequirementsTest() function (SystemRequirementsTest.cpp file).
- d. Report on code execution metrics.

The complete code listing for the DE Auto.cpp file may be found in Table 2.

**Table 2: DE Automatic Test Program****DE Automatic Test Program**

```
*****
*
* File: Auto.cpp
*
* Description: This file contains the AutoTest function where all simulation testing
* begins. This file is to be modified by the user to add
* automatic software tests.
*
*****
```

```
#include "Auto.h"

#include "Pointers.h"

#include "OScope.h"
#include "SigGen.h"

#include "SSGlobalData.h"

void SystemRequirementsTest();

void AutoTest()
{
    Message("Hello World");

    InitPointers(); // Init pointers to all parts for use in automatic testing

    CpuList[0] = (Cpu *)PPC_Cpu; // For debugger

    World *world = (World *)Sim::ExtNameToPartPtr("The World");

    char sysName[80];
    strcpy(sysName, world->GetSystemName());
    Message("System Name:");
    Message(sysName);

    // Generate "sim" files.
    //BOOL success = world->GenerateSimFiles(".\\GenSimFiles", TRUE);

    // Uncomment to generate pointers files
    //Sim::CreatePartPointerFiles();

    world->RestoreWindows("init.stu");
    world->SetAutoSpeed(FALSE);
    world->SetSpeed(12);

    pivot = 90.0;
    ang_elbow = 90.0;
    ang_shoulder = 90.0;
```



DE Automatic Test Program

```
ang_wristx = 90.0;
ang_wristy = 90.0;
ang_wristz = 90.0;
rot_wrist = 90.0;

eq.Delay(0.2);

Logic_Supply->SetPower(TRUE);

for(;;)
    eq.Delay(1.0);

// Set up data collection tasks
ExecutionMarkers *em;

// Find approximate start and end addresses for report generation
unsigned long start, end, loc;
start = PPC_Cpu->GetSymbolAddr("_start");
end = PPC_Cpu->GetSymbolAddr("__fini");

// Set up execution markers for percent coverage report
em = PPC_Cpu->GetPtrToExecutionMarkers();
em->CreateMarkers(start, end);
PPC_Cpu->SetExecutionMarkersEnable(TRUE);

// Clear "jump" markers for path coverage report
PPC_Cpu->GetPtrToJumpMarkers()->Clear();

// Record 0.8 seconds of subroutine and interrupt calls (and returns)
// for timing analysis
PPC_Cpu->SubroutineRecord(TRUE);
eq.Delay(0.3);
// careful, this can generate a huge file, so stop now
PPC_Cpu->SubroutineRecord(FALSE);

// Run System Requirements Test
SystemRequirementsTest();
eq.Delay(1.0);

// Save "jump" markers for path coverage report
PPC_Cpu->GetPtrToJumpMarkers()->Save("JumpMarks.txt");

int pcc, lc;
char buff[80];
FILE *execfp;

// Get data from execution markers
execfp = fopen("ExecutionData.txt", "w");
pcc = em->GetPercentCoverage();
sprintf(buff, "percent coverage: %d", pcc);
Message(buff);
fprintf(execfp, "Percent Coverage: %d\n", pcc);
```



DE Automatic Test Program

```
lc = em->GetLocationCount(PPC_Cpu->GetSymbolAddr("ControlProcess::PeriodicFunc(void)"));
sprintf(buff, "loc count: %d", lc);
Message(buff);
fprintf(execfp, "Loc Count at ControlProcess::PeriodicFunc(void): %d\n", lc);
em->Save("ExecMark.dat");
em->DestroyMarkers(); // Do this before creating markers in a different range
fclose(execfp);

// Dump a disassembly for general program reference
FILE *disa;
disa = fopen("Disassembly.txt", "w");
for(loc = start; loc < end; loc += 4)
    fprintf(disa, "%s\n", PPC_Cpu->Disassemble(loc));
fclose(disa);

Message("Done with cpu report tests");

//for(;;)
// eq.Delay(1.0);

// Interrupt Processing Reserve Test
// baddr1 is the break address at the start of periodic processing
// and baddr2 is approximately at the end
unsigned long baddr1, baddr2;
baddr1 = PPC_Cpu->GetSymbolAddr("ControlProcess::PeriodicFunc(void)");
PPC_Cpu->SetExtBreakpoint(0, baddr1); // set breakpoint 0
baddr2 = 0x70c8; // hard coded for now, should get value from stack
    // when the first breakpoint is hit
PPC_Cpu->SetExtBreakpoint(1, baddr2); // set breakpoint 1
PPC_Cpu->SetNumberOfExtBreakpoints(2);
PPC_Cpu->EnableBreak();

double t1, t2, t1last, int_period, int_proc_time;
unsigned long pc;
BOOL hit_bp1 = FALSE;
BOOL hit_bp2 = FALSE;

// initialize
while(!hit_bp1) {
    eq.DelayUntilBreak(1.0); // run the simulator until breakpoint
        // (1 sec timeout for breakpoint not hit)
    PPC_Cpu->ReadReg(96, &pc); // 96 is the code for the program counter
    if(pc == baddr1) {
        t1 = SimClock::GetSimTime();
        t1last = t1;
        hit_bp1 = TRUE;
    }
}

while(!hit_bp2) {
    eq.DelayUntilBreak(1.0); // run the simulator until breakpoint
        // (1 sec timeout for breakpoint not hit)
```



DE Automatic Test Program

```
PPC_Cpu->ReadReg(96, &pc);

if(pc == baddr2) {
    t2 = SimClock::GetSimTime();
    hit_bp2 = TRUE;
}
int_period = 0.0;
int_proc_time = 0.0;

for(int i = 0; i < 10; i++) {
    eq.DelayUntilBreak(1.0); // run the simulator until breakpoint
                            // (1 sec timeout for breakpoint not hit)
    PPC_Cpu->ReadReg(96, &pc);
    if(pc == baddr1) {
        t1 = SimClock::GetSimTime();
        int_period += t1 - t1last;
        t1last = t1;
    }
    else if(pc == baddr2) {
        t2 = SimClock::GetSimTime();
        int_proc_time += t2 - t1;
    }
}

int_period /= 5.0; // five times for each breakpoint
int_proc_time /= 5.0;
FILE *intfp = fopen("InterruptData.txt", "w");
fprintf(intfp, "Interrupt Period Average: %lf seconds\n", int_period);
fprintf(intfp, "Interrupt Processing Average: %lf seconds\n", int_proc_time);
fprintf(intfp, "Reserve Processing Percent: %lf pcnt\n",
        (int_period - int_proc_time) * 100.0 / int_period);
fclose(intfp);

Message("Done");

PPC_Cpu->DisableBreak(); // otherwise the cpu will keep stopping

for(;;)
    eq.Delay(1.0);
}
```



2.2 System Requirements Test Program

The SRMS system requirements tests are found in the SystemReqTest.cpp code listing in [Table 4](#). The 131 tests performed in this suite of tests covers only a subset of the total number of system requirements found in the ES. On a project slated for hardware development, all requirements would be tested. This same file is used for testing both the ES and the executable software running within the DE. If the MCDC code path coverage indicates untested paths, then either the code must be modified or additional tests written until all paths are exercised.

The system requirements tests fall into three basic groups:

- a. Fault reporting,
- b. RMA joint control, and
- c. Camera motor control.

2.2.1 Fault reporting tests

One of the required system behaviors specified in the ES is for AFDX devices to report their status with every message sent to the RMS computer. If any the status word of any device reports a fault, the computer lights the master Fault indicator on the RMS control panel. If any of the RMA joint motor controllers report a fault then the computer must also light the control panel “Err” indicator associated with that joint in the RMA Joint Status display. When all AFDX devices are indicating “No Fault” status, the computer must extinguish the respective fault lights on the control panel.

The fault reporting tests make use of the most basic of failure mode induction that can be accomplished with an IcoSim VSIL. Each AFDX device (except the camera lamps) reports its built-in-test (BIT) status along with any data requested. No actual BIT has been implemented for this project so the status is always reported as “No Fault” during simulator operation. However, the AFDX device parts have been created with two functions that support testing the requirements – one that allows setting the part status word to any value, and one that allows clearing the part status word.

Under sequential test control, the status word of each AFDX device is set to a specified failure condition code. After allowing for the fault to propagate through the system the state of the control panel lights is read and compared with the expected value for a pass/fail determination. The device status word is then cleared (“No Fault” state) and after the appropriate delay the control panel display is checked to see if the appropriate indicators have been extinguished.

In order to set the device status, one must first get a pointer to the desired instance of the part by finding it in pointers.cpp, a file that is automatically generated when a simulator build is created. Then simply send the desired fault code to the function within that part instance. [Table 3](#) gives an example of how it’s done for the RMA shoulder yaw motor controller:

Table 3: Example of Fault Reporting Verification Test

Example of Fault Reporting Verification Test

```
// Test RMA Shoulder Yaw motor fault reporting
sprintf(resfp, "\nTest Number %d: ", testnum++);
sprintf(resfp, "Shoulder Yaw Motor Fault - SET\n");

Yaw_Controller_3->SetStatus((int)MotorFault);
eq.Delay(1.2);
```



Example of Fault Reporting Verification Test

```
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == (shldYawFault | MasterFault))
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Shoulder Yaw Motor Fault - CLEAR\n");

// Test RMA Shoulder Yaw motor fault clearing
Yaw_Controller_3->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");
```

While there is only one motor controller part used in the SRMS, there are 21 motors that use the part and correspondingly 21 instances of the motor controller part. As you can see in the example above, Yaw_Controller_3 points to the RMA shoulder yaw motor controller. Motor controller status is manipulated by writing to the “SetStatus” and “ClearStatus” functions. The state of the control panel indicators is read through the “GetControlPanelData” function in the RMS Control Panel part.

2.2.2 RMA joint control tests

The RMA joints are moved individually using command inputs from the control panel. Manual control mode allows the operator to increment or decrement each joint and then stop at the press (click) of the “STOP” button. Angle seek mode allows the operator to input a destination angle to which the selected joint is intended to move. To test these functions, the control panel buttons are driven under test control and the joint velocity and position are monitored and compared with expected performance values.

2.2.3 Camera motor control tests

Each of the three camera motors is controlled manually through buttons found to the right of the control panel video display monitors. The tests in this section are designed to confirm that the motors move with the correct velocity corresponding to what is expected for the button that is clicked under test control.

**Table 4: SRMS System Requirements Test Program****System Requirements Tests**

```
*****
*
* File: SystemReqTest.cpp
*
* Description: This file contains a subset of the System Requirements Test.
*               This test should run on the ES and DE simulators unchanged.
*
*****
```

```
#include "Pointers.h"
#include "ControlPanel_ICD.h"

enum {
    ANGLE_SEEK_MODE,
    MANUAL_MODE
};

// Motor controller status bits
enum MotorStatusIDs {
    MotorFault = 1,
    ControllerFault = 2,
    NobodysFault = 4,
    YourFault = 8,
    MyFault = 0x10,
    SanAndreasFault = 0x20,
    TowerFawlt = 0x40,
};

// Data module status bits
enum DataModStatusIDs {
    ResistanceFault = 1,
    DataModuleFault = 2,
};

// Image sensor status bits
enum ImageSensorStatusIDs {
    SensorFault = 1,
    Tarnished = 2,
    Faded = 4,
    LargerThanLife = 8,
};

char *motorStr[21] = {

    "Shoulder Yaw",
    "Shoulder Pitch",
    "Elbow Pitch",
    "Wrist Pitch",
```



System Requirements Tests

```
"Wrist Yaw",
"Wrist Roll",

"UpperArmCam Yaw",
"UpperArmCam Pitch",
"UpperArmCam Zoom",
"LowerArmCam Yaw",
"LowerArmCam Pitch",
"LowerArmCam Zoom",
"WristCam Yaw",
"WristCam Pitch",
"WristCam Zoom",
"FwdBayCam Yaw",
"FwdBayCam Pitch",
"FwdBayCam Zoom",
"AftBayCam Yaw",
"AftBayCam Pitch",
"AftBayCam Zoom",
};

void SystemRequirementsTest()
{
    double display_angle;
    double display_vel;
    double expected_vel;
    double actual_vel;
    double err_range;
    double start_angle;
    double expected_angle;
    double actual_angle;

    PlanetaryGearSet *gears[21] = {
        Gear_8,
        Gear_7,
        Gear_12,
        Gear_16,
        Gear_17,
        Gear_18,

        Gear_10,
        Gear_9,
        Gear_11,
        Gear_14,
        Gear_13,
        Gear_15,
        Gear_20,
        Gear_19,
        Gear_21,
        Gear_2,
        Gear_1,
        Gear_3,
        Gear_5,
```



System Requirements Tests

```
Gear_4,  
Gear_6,  
};  
  
FILE *resfp;  
  
resfp = fopen("SysReqTest.res", "w");  
  
eq.Delay(5.0); // initial start up delay  
  
int testnum = 1;  
  
// Fault reporting tests  
  
// For each AFDX device do the following tests:  
// 1. Force device internal status to a valid fault value  
// 2. Verify that fault is displayed correctly on RMS Control Panel  
// 3. Force device to clear internal status value to "No Fault" state  
// 4. Verify that RMS Control Panel displayed fault is automatically cleared  
  
int mot, mod, cmd, vid1, vid2, flt;  
  
fprintf(resfp, "\n*****\n");  
fprintf(resfp, "*      AFDX Device Fault Reporting & Auto Clearing Tests      *\n");  
fprintf(resfp, "*****\n\n");  
  
// Test RMA Shoulder Yaw motor fault reporting  
fprintf(resfp, "\nTest Number %d: ", testnum++);  
fprintf(resfp, "Shoulder Yaw Motor Fault - SET\n");  
  
Yaw_Controller_3->SetStatus((int)MotorFault);  
eq.Delay(1.2);  
  
// Retrieve control panel indicator state  
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);  
if(flt == (shldYawFault | MasterFault))  
    fprintf(resfp, "**** PASSED\n");  
else  
    fprintf(resfp, "\n**** FAILED ****\n");  
  
fprintf(resfp, "\nTest Number %d: ", testnum++);  
fprintf(resfp, "Shoulder Yaw Motor Fault - CLEAR\n");  
  
// Test RMA Shoulder Yaw motor fault clearing  
Yaw_Controller_3->ClearStatus();  
eq.Delay(1.2);  
// Retrieve control panel indicator state  
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);  
if(flt == 0)  
    fprintf(resfp, "**** PASSED\n");
```



System Requirements Tests

```
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test RMA Shoulder Pitch motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Shoulder Pitch Motor Fault - SET\n");

Pitch_Controller_3->SetStatus((int)MotorFault);
eq.Delay(1.2);

// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == (shldPitchFault | MasterFault))
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test RMA Shoulder Pitch motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Shoulder Pitch Motor Fault - CLEAR\n");

Pitch_Controller_3->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test RMA Elbow motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Elbow Pitch Motor Fault - SET\n");

Pitch_Controller_5->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == (elbowPitchFault | MasterFault))
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test RMA Elbow motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Elbow Pitch Motor Fault - CLEAR\n");

Pitch_Controller_5->ClearStatus();
```



System Requirements Tests

```
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test RMA wrist pitch motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Wrist Pitch Motor Fault - SET\n");

Pitch_Controller_7->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == (wristPitchFault | MasterFault))
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test RMA wrist pitch motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Wrist Pitch Motor Fault - CLEAR\n");

Pitch_Controller_7->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test RMA Wrist Yaw motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Wrist Yaw Motor Fault - SET\n");

Yaw_Controller_6->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == (wristYawFault | MasterFault))
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test RMA wrist yaw motor fault clearing
```



System Requirements Tests

```
fprintf(resfp, "\nTest Number %d: ", testnum++);  
fprintf(resfp, "Wrist Yaw Motor Fault - CLEAR\n");  
  
Yaw_Controller_6->ClearStatus();  
eq.Delay(1.2);  
// Retrieve control panel indicator state  
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);  
if(flt == 0)  
    fprintf(resfp, "**** PASSED\n");  
else  
    fprintf(resfp, "\n**** FAILED ****\n");  
  
  
// Test RMA wrist roll motor fault reporting  
fprintf(resfp, "\nTest Number %d: ", testnum++);  
fprintf(resfp, "Wrist Roll Motor Fault - SET\n");  
  
Roll_Controller->SetStatus((int)MotorFault);  
eq.Delay(1.2);  
// Retrieve control panel indicator state  
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);  
if(flt == (wristsRollFault | MasterFault))  
    fprintf(resfp, "**** PASSED\n");  
else  
    fprintf(resfp, "\n**** FAILED ****\n");  
  
// Test RMA wrist roll motor fault clearing  
fprintf(resfp, "\nTest Number %d: ", testnum++);  
fprintf(resfp, "Wrist Roll Motor Fault - CLEAR\n");  
  
Roll_Controller->ClearStatus();  
eq.Delay(1.2);  
// Retrieve control panel indicator state  
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);  
if(flt == 0)  
    fprintf(resfp, "**** PASSED\n");  
else  
    fprintf(resfp, "\n**** FAILED ****\n");  
  
  
// Test Forward Bay camera pitch motor fault reporting  
fprintf(resfp, "\nTest Number %d: ", testnum++);  
fprintf(resfp, "Fwd Bay Cam Pitch Motor Fault - SET\n");  
  
Pitch_Controller_1->SetStatus((int)MotorFault);  
eq.Delay(1.2);  
// Retrieve control panel indicator state  
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);  
if(flt == MasterFault)  
    fprintf(resfp, "**** PASSED\n");
```



System Requirements Tests

```
else
    fprintf(resfp, "\n***** FAILED *****\n");

// Test Forward Bay camera pitch motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Fwd Bay Cam Pitch Motor Fault - CLEAR\n");

Pitch_Controller_1->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "***** PASSED\n");
else
    fprintf(resfp, "\n***** FAILED *****\n");

// Test Forward Bay camera yaw motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Fwd Bay Cam Yaw Motor Fault - SET\n");

Yaw_Controller_1->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "***** PASSED\n");
else
    fprintf(resfp, "\n***** FAILED *****\n");

// Test Forward Bay camera yaw motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Fwd Bay Cam Yaw Motor Fault - CLEAR\n");

Yaw_Controller_1->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "***** PASSED\n");
else
    fprintf(resfp, "\n***** FAILED *****\n");

// Test Forward Bay camera zoom motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Fwd Bay Cam Zoom Motor Fault - SET\n");
```



System Requirements Tests

```
Zoom_Controller_1->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "***** PASSED\n");
else
    fprintf(resfp, "\n***** FAILED *****\n");

// Test Forward Bay camera zoom motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Fwd Bay Cam Zoom Motor Fault - CLEAR\n");

Zoom_Controller_1->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "***** PASSED\n");
else
    fprintf(resfp, "\n***** FAILED *****\n");

// Test Aft Bay camera pitch motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Aft Bay Cam Pitch Motor Fault - SET\n");

Pitch_Controller_2->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "***** PASSED\n");
else
    fprintf(resfp, "\n***** FAILED *****\n");

// Test Aft Bay camera pitch motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Aft Bay Cam Pitch Motor Fault - CLEAR\n");

Pitch_Controller_2->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "***** PASSED\n");
else
    fprintf(resfp, "\n***** FAILED *****\n");
```



System Requirements Tests

```
// Test Aft Bay camera yaw motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Aft Bay Cam Yaw Motor Fault - SET\n");

Yaw_Controller_2->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Aft Bay camera yaw motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Aft Bay Cam Yaw Motor Fault - CLEAR\n");

Yaw_Controller_2->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Aft Bay camera zoom motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Aft Bay Cam Zoom Motor Fault - SET\n");

Zoom_Controller_2->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Aft Bay camera zoom motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Aft Bay Cam Zoom Motor Fault - CLEAR\n");

Zoom_Controller_2->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
```



System Requirements Tests

```
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Upper Arm camera pitch motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "UA Cam Pitch Motor Fault - SET\n");

Pitch_Controller_4->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Upper Arm camera pitch motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "UA Cam Pitch Motor Fault - CLEAR\n");

Pitch_Controller_4->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Upper Arm camera yaw motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "UA Cam Yaw Motor Fault - SET\n");

Yaw_Controller_4->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Upper Arm camera yaw motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
```



System Requirements Tests

```
fprintf(resfp, "UA Cam Yaw Motor Fault - CLEAR\n");

Yaw_Controller_4->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "***** PASSED\n");
else
    fprintf(resfp, "\n***** FAILED *****\n");

// Test Upper Arm camera zoom motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "UA Cam Zoom Motor Fault - SET\n");

Zoom_Controller_3->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "***** PASSED\n");
else
    fprintf(resfp, "\n***** FAILED *****\n");

// Test Upper Arm camera zoom motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "UA Cam Zoom Motor Fault - CLEAR\n");

Zoom_Controller_3->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "***** PASSED\n");
else
    fprintf(resfp, "\n***** FAILED *****\n");

// Test Lower Arm camera pitch motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "LA Cam Pitch Motor Fault - SET\n");

Pitch_Controller_6->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
```



System Requirements Tests

```
fprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Test Lower Arm camera pitch motor fault clearing
sprintf(resfp, "\nTest Number %d: ", testnum++);
sprintf(resfp, "LA Cam Pitch Motor Fault - CLEAR\n");

Pitch_Controller_6->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Test Lower Arm camera yaw motor fault reporting
sprintf(resfp, "\nTest Number %d: ", testnum++);
sprintf(resfp, "LA Cam Yaw Motor Fault - SET\n");

Yaw_Controller_5->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Test Lower Arm camera yaw motor fault clearing
sprintf(resfp, "\nTest Number %d: ", testnum++);
sprintf(resfp, "LA Cam Yaw Motor Fault - CLEAR\n");

Yaw_Controller_5->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Test Lower Arm camera zoom motor fault reporting
sprintf(resfp, "\nTest Number %d: ", testnum++);
sprintf(resfp, "LA Cam Zoom Motor Fault - SET\n");
```



System Requirements Tests

```
Zoom_Controller_4->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Lower Arm camera zoom motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "LA Cam Zoom Motor Fault - CLEAR\n");

Zoom_Controller_4->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Wrist camera pitch motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Wrist Cam Pitch Motor Fault - SET\n");

Pitch_Controller_8->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Wrist camera pitch motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Wrist Cam Pitch Motor Fault - CLEAR\n");

Pitch_Controller_8->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");
```



System Requirements Tests

```
// Test Wrist camera yaw motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Wrist Cam Yaw Motor Fault - SET\n");

Yaw_Controller_7->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Wrist camera yaw motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Wrist Cam Yaw Motor Fault - CLEAR\n");

Yaw_Controller_7->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");



// Test Wrist camera zoom motor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Wrist Cam Zoom Motor Fault - SET\n");

Zoom_Controller_5->SetStatus((int)MotorFault);
eq.Delay(1.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test Wrist camera zoom motor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Wrist Cam Zoom Motor Fault - CLEAR\n");

Zoom_Controller_5->ClearStatus();
eq.Delay(1.2);
// Retrieve control panel indicator state
```



System Requirements Tests

```
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Test UA strain gauge data module fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Upper Arm Strain Gauge Data Module Fault - SET\n");

Data_Module_1->SetStatus((int)ResistanceFault);
eq.Delay(0.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Test UA strain gauge data module fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Upper Arm Strain Gauge Data Module Fault - CLEAR\n");

Data_Module_1->ClearStatus();
eq.Delay(0.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Test LA strain gauge data module fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Lower Arm Strain Gauge Data Module Fault - SET\n");

Data_Module_2->SetStatus((int)ResistanceFault);
eq.Delay(0.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Test LA strain gauge data module fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
```



System Requirements Tests

```
fprintf(resfp, "Lower Arm Strain Gauge Data Module Fault - CLEAR\n");
```

```
Data_Module_2->ClearStatus();
eq.Delay(0.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");
```

```
// Test Fwd Bay Camera image sensor fault reporting
```

```
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Fwd Bay Camera image sensor Fault - SET\n");
```

```
ImageSensor__1->SetStatus((int)SensorFault);
eq.Delay(0.2);
```

```
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");
```

```
// Test Fwd Bay Camera image sensor fault clearing
```

```
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Fwd Bay Camera image sensor Fault - CLEAR\n");
```

```
ImageSensor__1->ClearStatus();
eq.Delay(0.2);
```

```
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");
```

```
// Test Aft Bay Camera image sensor fault reporting
```

```
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Aft Bay Camera image sensor Fault - SET\n");
```

```
ImageSensor__2->SetStatus((int)SensorFault);
eq.Delay(0.2);
```

```
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "**** PASSED\n");
else
```



System Requirements Tests

```
fprintf(resfp, "\n**** FAILED ****\n");

// Test Aft Bay Camera image sensor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Aft Bay Camera image sensor Fault - CLEAR\n");

ImageSensor__2->ClearStatus();
eq.Delay(0.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test UA Camera image sensor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "UA Camera image sensor Fault - SET\n");

ImageSensor__3->SetStatus((int)SensorFault);
eq.Delay(0.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test UA Camera image sensor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "UA Camera image sensor Fault - CLEAR\n");

ImageSensor__3->ClearStatus();
eq.Delay(0.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

// Test LA Camera image sensor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "LA Camera image sensor Fault - SET\n");

ImageSensor__4->SetStatus((int)SensorFault);
eq.Delay(0.2);
// Retrieve control panel indicator state
```



System Requirements Tests

```
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Test LA Camera image sensor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "LA Camera image sensor Fault - CLEAR\n");

ImageSensor_4->ClearStatus();
eq.Delay(0.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Test Wrist Camera image sensor fault reporting
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Wrist Camera image sensor Fault - SET\n");

ImageSensor_5->SetStatus((int)SensorFault);
eq.Delay(0.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == MasterFault)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Test Wrist Camera image sensor fault clearing
fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Wrist Camera image sensor Fault - CLEAR\n");

ImageSensor_5->ClearStatus();
eq.Delay(0.2);
// Retrieve control panel indicator state
RMS_Ctrl_Panel->GetControlPanelData(mot, mod, cmd, vid1, vid2, flt);
if(flt == 0)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

// Joint motor tests
```



System Requirements Tests

```
// for each joint motor in the system
// 1. Command Manual Mode
// 2. Command Increase Angle
// 3. Verify actual velocity against expected
// 4. Verify actual velocity against displayed
// 5. Command Stop
// 6. Verify stop
// 7. Command Decrease Angle
// 8. Verify actual velocity against expected
// 9. Verify actual velocity against displayed
// 10. Command Stop
// 11. Verify stop
// 12. Command Angle Seek Mode
// 13. Command an angle greater than current position
// 14. Verify actual velocity against expected
// 15. Verify actual velocity against displayed
// 16. Wait for stop
// 17. Verify final angle against commanded angle.
// 18. Command an angle less than than current position
// 19. Verify actual velocity against expected
// 20. Verify actual velocity against displayed
// 21. Wait for stop
// 22. Verify final angle against commanded angle.

fprintf(resfp, "\n\n*****\n");
fprintf(resfp, "* RMA Joint Control Tests *\n");
fprintf(resfp, "*****\n\n");

for(mot = 0; mot < NumRMAJoints; mot++) { // There are six RMA motors

    fprintf(resfp, "Testing %s Control\n\n", motorStr[mot]);

    RMS_Ctrl_Panel->SetMode(MANUAL_MODE);
    eq.Delay(0.2);
    RMS_Ctrl_Panel->SetMotor(mot);
    eq.Delay(0.2);

    fprintf(resfp, "\nTest Number %d: ", testnum++);
    fprintf(resfp, "Positive Velocity\n");

    RMS_Ctrl_Panel->SetCommand(INC);
    eq.Delay(5.0);

    actual_vel = gears[mot]->GetLowSpeedVelocity() * 180.0 / pi;
    display_angle = RMS_Ctrl_Panel->GetMotorData(mot, 0);
    display_vel = RMS_Ctrl_Panel->GetMotorData(mot, 2);
    expected_vel = (600.0 / 500.0) * (1.0 / 60.0) * 360.0;

    fprintf(resfp, " Actual value: %lf\n", actual_vel);
    fprintf(resfp, " Display value: %lf\n", display_vel);
    fprintf(resfp, " Expected value: %lf\n", expected_vel);
    err_range = 2.0;
```



System Requirements Tests

```
fprintf(resfp, " Error range: +- %lf\n", err_range);
fprintf(resfp, "Verify Actual Value\n");
if(fabs(actual_vel - expected_vel) < err_range)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");
fprintf(resfp, "Verify Display Value\n");
if(fabs(display_vel - expected_vel) < err_range)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

RMS_Ctrl_Panel->SetCommand(STOP);

fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Zero Velocity\n");

eq.Delay(3.0);

actual_vel = gears[mot]->GetLowSpeedVelocity() * 180.0 / pi;
display_angle = RMS_Ctrl_Panel->GetMotorData(mot, 0);
display_vel = RMS_Ctrl_Panel->GetMotorData(mot, 2);
expected_vel = 0.0;

fprintf(resfp, " Actual value: %lf\n", actual_vel);
fprintf(resfp, " Display value: %lf\n", display_vel);
fprintf(resfp, " Expected value: %lf\n", expected_vel);
err_range = 2.0;
fprintf(resfp, " Error range: +- %lf\n", err_range);
fprintf(resfp, "Verify Actual Value\n");
if(fabs(actual_vel - expected_vel) < err_range)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");
fprintf(resfp, "Verify Display Value\n");
if(fabs(display_vel - expected_vel) < err_range)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Negative Velocity\n");

RMS_Ctrl_Panel->SetCommand(DEC);
eq.Delay(5.0);

actual_vel = gears[mot]->GetLowSpeedVelocity() * 180.0 / pi;
display_angle = RMS_Ctrl_Panel->GetMotorData(mot, 0);
display_vel = RMS_Ctrl_Panel->GetMotorData(mot, 2);
expected_vel = -(600.0 / 500.0) * (1.0 / 60.0) * 360.0;

fprintf(resfp, " Actual value: %lf\n", actual_vel);
```



System Requirements Tests

```
fprintf(resfp, " Display value: %lf\n", display_vel);
fprintf(resfp, " Expected value: %lf\n", expected_vel);
err_range = 2.0;
fprintf(resfp, " Error range: +- %lf\n", err_range);
fprintf(resfp, "Verify Actual Value\n");
if(fabs(actual_vel - expected_vel) < err_range)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");
fprintf(resfp, "Verify Display Value\n");
if(fabs(display_vel - expected_vel) < err_range)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");
/*
RMS_Ctrl_Panel->SetCommand(STOP);
eq.Delay(3.0);
eq.Delay(1.0);
*/
RMS_Ctrl_Panel->SetMode(ANGLE_SEEK_MODE); //Changing modes stops all RMA joints
eq.Delay(2.0);

fprintf(resfp, "\nTest Number %d:", testnum++);
fprintf(resfp, "Positive Angle Seek\n");

start_angle = gears[mot]->GetLowSpeedAngle() * 180.0 / pi;
RMS_Ctrl_Panel->SetAngle((int)(start_angle + 20.0));
eq.Delay(10.0);

actual_angle = gears[mot]->GetLowSpeedAngle() * 180.0 / pi;
display_angle = RMS_Ctrl_Panel->GetMotorData(mot, 0);
expected_angle = start_angle + 20.0;

fprintf(resfp, " Actual angle: %lf\n", actual_angle);
fprintf(resfp, " Display angle: %lf\n", display_angle);
fprintf(resfp, " Expected angle: %lf\n", expected_angle);
err_range = 2.5;
fprintf(resfp, " Error range: +- %lf\n", err_range);
fprintf(resfp, "Verify Actual Angle\n");
if(fabs(actual_angle - expected_angle) < err_range)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");
fprintf(resfp, "Verify Display Angle\n");
if(fabs(display_angle - expected_angle) < err_range)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

fprintf(resfp, "\nTest Number %d: ", testnum++);
fprintf(resfp, "Negative Angle Seek\n");
```



System Requirements Tests

```
start_angle = gears[mot]->GetLowSpeedAngle() * 180.0 / pi;
RMS_Ctrl_Panel->SetAngle((int)(start_angle - 20.0));
eq.Delay(10.0);

actual_angle = gears[mot]->GetLowSpeedAngle() * 180.0 / pi;
display_angle = RMS_Ctrl_Panel->GetMotorData(mot, 0);
expected_angle = start_angle - 20.0;

fprintf(resfp, " Actual angle: %lf\n", actual_angle);
fprintf(resfp, " Display angle: %lf\n", display_angle);
fprintf(resfp, " Expected angle: %lf\n", expected_angle);
err_range = 2.5;
fprintf(resfp, " Error range: +- %lf\n", err_range);
fprintf(resfp, "Verify Actual Angle\n");
if(fabs(actual_angle - expected_angle) < err_range)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");
fprintf(resfp, "Verify Display Angle\n");
if(fabs(display_angle - expected_angle) < err_range)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

fprintf(resfp, "\n\n*****\n\n");
```

}

```
eq.Delay(5.0);
```

// Camera motor tests

```
// for each camera motor in the system
// 1. Command Increase Angle
// 2. Verify actual velocity against expected
// 3. Command Stop
// 4. Verify stop
// 5. Command Decrease Angle
// 6. Verify actual velocity against expected
// 7. Command Stop
```

```
fprintf(resfp, "\n\n*****\n\n");
fprintf(resfp, "          *           Camera Motor Control Tests           *\n");
fprintf(resfp, "          *****\n\n");
```

```
for(mot = 6; mot < 21; mot++) { // There are 15 camera motors
    fprintf(resfp, "Testing %s Control\n\n", motorStr[mot]);
```



System Requirements Tests

```
RMS_Ctrl_Panel->SetVideo1Source((mot - 6) / 3);
eq.Delay(0.2);

sprintf(resfp, "\nTest Number %d: ", testnum++);
sprintf(resfp, "Positive Velocity\n");

RMS_Ctrl_Panel->SetCam1ControlParams((mot - 6) % 3, 1); // start

eq.Delay(5.0);

actual_vel = gears[mot]->GetLowSpeedVelocity() * 180.0 / pi;
expected_vel = (600.0 / 500.0) * (1.0 / 60.0) * 360.0;

sprintf(resfp, " Actual value: %lf\n", actual_vel);
sprintf(resfp, " Expected value: %lf\n", expected_vel);
err_range = 2.0;
sprintf(resfp, " Error range: +- %lf\n", err_range);
sprintf(resfp, "Verify Actual Value\n");
if(fabs(actual_vel - expected_vel) < err_range)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

RMS_Ctrl_Panel->SetCam1ControlParams((mot - 6) % 3, 1); // stop

sprintf(resfp, "\nTest Number %d: ", testnum++);
sprintf(resfp, "Zero Velocity\n");

eq.Delay(2.0);

actual_vel = gears[mot]->GetLowSpeedVelocity() * 180.0 / pi;
expected_vel = 0.0;

sprintf(resfp, " Actual value: %lf\n", actual_vel);
sprintf(resfp, " Expected value: %lf\n", expected_vel);
err_range = 2.0;
sprintf(resfp, " Error range: +- %lf\n", err_range);
sprintf(resfp, "Verify Actual Value\n");
if(fabs(actual_vel - expected_vel) < err_range)
    sprintf(resfp, "**** PASSED\n");
else
    sprintf(resfp, "\n**** FAILED ****\n");

sprintf(resfp, "\nTest Number %d: ", testnum++);
sprintf(resfp, "Negative Velocity\n");

RMS_Ctrl_Panel->SetCam1ControlParams((mot - 6) % 3, 0); // start
eq.Delay(5.0);

actual_vel = gears[mot]->GetLowSpeedVelocity() * 180.0 / pi;
expected_vel = -(600.0 / 500.0) * (1.0 / 60.0) * 360.0;
```



System Requirements Tests

```
fprintf(resfp, " Actual value: %lf\n", actual_vel);
fprintf(resfp, " Expected value: %lf\n", expected_vel);
err_range = 2.0;
fprintf(resfp, " Error range: +- %lf\n", err_range);
fprintf(resfp, "Verify Actual Value\n");
if(fabs(actual_vel - expected_vel) < err_range)
    fprintf(resfp, "**** PASSED\n");
else
    fprintf(resfp, "\n**** FAILED ****\n");

RMS_Ctrl_Panel->SetCam1ControlParams((mot - 6) % 3, 0); // stop

fprintf(resfp, "\n\n*****\n");
fclose(resfp);

eq.Delay(3.0);

Message("Test Complete");
}
```